**Importing the Library:**

from \_\_future\_\_ import print\_function

import keras

from keras.datasets import cifar10

from keras.preprocessing.image import ImageDataGenerator

from keras.models import Sequential

from keras.layers import Dense, Dropout, Activation, Flatten

from keras.layers import Conv2D, MaxPooling2D

from keras.models import load\_model

import os

batch\_size = 32

num\_classes = 10

epochs = 1

# Loads the CIFAR dataset

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

**Loading the dataset:**

# Get filenames in list

from os import listdir

from os.path import isfile, join

mypath = "./datasets/images/"

file\_names = [f for f in listdir(mypath) if isfile(join(mypath, f))]

print(str(len(file\_names)) + ' images loaded')

**Script to create dataset from messy dataset:**

import cv2

import numpy as np

import sys

import os

import shutil

# Extract 1000 for our training data and 500 for our validation set

# Takes about ~20 seconds to run

dog\_count = 0

cat\_count = 0

training\_size = 1000

test\_size = 500

training\_images = []

training\_labels = []

test\_images = []

test\_labels = []

size = 150

dog\_dir\_train = "./datasets/catsvsdogs/train/dogs/"

cat\_dir\_train = "./datasets/catsvsdogs/train/cats/"

dog\_dir\_val = "./datasets/catsvsdogs/validation/dogs/"

cat\_dir\_val = "./datasets/catsvsdogs/validation/cats/"

def make\_dir(directory):

if os.path.exists(directory):

shutil.rmtree(directory)

os.makedirs(directory)

make\_dir(dog\_dir\_train)

make\_dir(cat\_dir\_train)

make\_dir(dog\_dir\_val)

make\_dir(cat\_dir\_val)

def getZeros(number):

if(number > 10 and number < 100):

return "0"

if(number < 10):

return "00"

else:

return ""

for i, file in enumerate(file\_names):

if file\_names[i][0] == "d":

dog\_count += 1

image = cv2.imread(mypath+file)

image = cv2.resize(image, (size, size), interpolation = cv2.INTER\_AREA)

if dog\_count <= training\_size:

training\_images.append(image)

training\_labels.append(1)

zeros = getZeros(dog\_count)

cv2.imwrite(dog\_dir\_train + "dog" + str(zeros) + str(dog\_count) + ".jpg", image)

if dog\_count > training\_size and dog\_count <= training\_size+test\_size:

test\_images.append(image)

test\_labels.append(1)

zeros = getZeros(dog\_count-1000)

cv2.imwrite(dog\_dir\_val + "dog" + str(zeros) + str(dog\_count-1000) + ".jpg", image)

if file\_names[i][0] == "c":

cat\_count += 1

image = cv2.imread(mypath+file)

image = cv2.resize(image, (size, size), interpolation = cv2.INTER\_AREA)

if cat\_count <= training\_size:

training\_images.append(image)

training\_labels.append(0)

zeros = getZeros(cat\_count)

cv2.imwrite(cat\_dir\_train + "cat" + str(zeros) + str(cat\_count) + ".jpg", image)

if cat\_count > training\_size and cat\_count <= training\_size+test\_size:

test\_images.append(image)

test\_labels.append(0)

zeros = getZeros(cat\_count-1000)

cv2.imwrite(cat\_dir\_val + "cat" + str(zeros) + str(cat\_count-1000) + ".jpg", image)

if dog\_count == training\_size+test\_size and cat\_count == training\_size+test\_size:

break

print("Training and Test Data Extraction Complete")

# Using numpy's savez function to store our loaded data as NPZ files

np.savez('cats\_vs\_dogs\_training\_data.npz', np.array(training\_images))

np.savez('cats\_vs\_dogs\_training\_labels.npz', np.array(training\_labels))

np.savez('cats\_vs\_dogs\_test\_data.npz', np.array(test\_images))

np.savez('cats\_vs\_dogs\_test\_labels.npz', np.array(test\_labels))

# Loader Function

import numpy as np

def load\_data\_training\_and\_test(datasetname):

npzfile = np.load(datasetname + "\_training\_data.npz")

train = npzfile['arr\_0']

npzfile = np.load(datasetname + "\_training\_labels.npz")

train\_labels = npzfile['arr\_0']

npzfile = np.load(datasetname + "\_test\_data.npz")

test = npzfile['arr\_0']

npzfile = np.load(datasetname + "\_test\_labels.npz")

test\_labels = npzfile['arr\_0']

return (train, train\_labels), (test, test\_labels)

(x\_train, y\_train), (x\_test, y\_test) = load\_data\_training\_and\_test("cats\_vs\_dogs")

# Reshaping our label data from (2000,) to (2000,1) and test data from (1000,) to (1000,1)

y\_train = y\_train.reshape(y\_train.shape[0], 1)

y\_test = y\_test.reshape(y\_test.shape[0], 1)

# Change our image type to float32 data type

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

# Normalize our data by changing the range from (0 to 255) to (0 to 1)

x\_train /= 255

x\_test /= 255

print(x\_train.shape)

print(y\_train.shape)

print(x\_test.shape)

print(y\_test.shape)

**Data Augmentation:**

import os

import numpy as np

from keras.models import Sequential

from keras.layers import Activation, Dropout, Flatten, Dense

from keras.preprocessing.image import ImageDataGenerator

from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D

from keras import optimizers

import scipy

import pylab as pl

import matplotlib.cm as cm

%matplotlib inline

input\_shape = (150, 150, 3)

img\_width = 150

img\_height = 150

nb\_train\_samples = 2000

nb\_validation\_samples = 1000

batch\_size = 16

epochs = 25

train\_data\_dir = './datasets/catsvsdogs/train'

validation\_data\_dir = './datasets/catsvsdogs/validation'

# Creating our data generator for our test data

validation\_datagen = ImageDataGenerator(

# used to rescale the pixel values from [0, 255] to [0, 1] interval

rescale = 1./255)

# Creating our data generator for our training data

train\_datagen = ImageDataGenerator(

rescale = 1./255, # normalize pixel values to [0,1]

rotation\_range = 30, # randomly applies rotations

width\_shift\_range = 0.3, # randomly applies width shifting

height\_shift\_range = 0.3, # randomly applies height shifting

horizontal\_flip = True, # randonly flips the image

fill\_mode = 'nearest') # uses the fill mode nearest to fill gaps created by the above

# Specify criteria about our training data, such as the directory, image size, batch size and type

# automagically retrieve images and their classes for train and validation sets

train\_generator = train\_datagen.flow\_from\_directory(

train\_data\_dir,

target\_size = (img\_width, img\_height),

batch\_size = batch\_size,

class\_mode = 'binary',

shuffle = True)

validation\_generator = validation\_datagen.flow\_from\_directory(

validation\_data\_dir,

target\_size = (img\_width, img\_height),

batch\_size = batch\_size,

class\_mode = 'binary',

shuffle = False)

history = model.fit\_generator(

train\_generator,

steps\_per\_epoch = nb\_train\_samples // batch\_size,

epochs = epochs,

validation\_data = validation\_generator,

validation\_steps = nb\_validation\_samples // batch\_size)

# Display our data shape/dimensions

print('x\_train shape:', x\_train.shape)

print(x\_train.shape[0], 'train samples')

print(x\_test.shape[0], 'test samples')

# Format our training data by Normalizing and changing data type

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

x\_train /= 255

x\_test /= 255

# Now we one hot encode outputs

y\_train = keras.utils.to\_categorical(y\_train, num\_classes)

y\_test = keras.utils.to\_categorical(y\_test, num\_classes)

=====================================================================================

**Plotting the dataset:**

**# importing matplot lib**

**import matplotlib.pyplot as plt**

**# Plots 6 images, note subplot's arugments are nrows,ncols,index**

**# we set the color map to grey since our image dataset is grayscale**

**plt.subplot(331)**

**random\_num = np.random.randint(0,len(x\_train))**

**plt.imshow(x\_train[random\_num], cmap=plt.get\_cmap('gray'))**

**plt.subplot(332)**

**random\_num = np.random.randint(0,len(x\_train))**

**plt.imshow(x\_train[random\_num], cmap=plt.get\_cmap('gray'))**

**plt.subplot(333)**

**random\_num = np.random.randint(0,len(x\_train))**

**plt.imshow(x\_train[random\_num], cmap=plt.get\_cmap('gray'))**

**plt.subplot(334)**

**random\_num = np.random.randint(0,len(x\_train))**

**plt.imshow(x\_train[random\_num], cmap=plt.get\_cmap('gray'))**

**plt.subplot(335)**

**random\_num = np.random.randint(0,len(x\_train))**

**plt.imshow(x\_train[random\_num], cmap=plt.get\_cmap('gray'))**

**plt.subplot(336)**

**random\_num = np.random.randint(0,len(x\_train))**

**plt.imshow(x\_train[random\_num], cmap=plt.get\_cmap('gray'))**

**# Display out plots**

**plt.show()**

**Making the Model:**

model = Sequential()

# Padding = 'same' results in padding the input such that

# the output has the same length as the original input

model.add(Conv2D(32, (3, 3), padding='same',

input\_shape=x\_train.shape[1:]))

model.add(Activation('relu'))

model.add(Conv2D(32, (3, 3)))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))

model.add(Activation('relu'))

model.add(Conv2D(64, (3, 3)))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(512))

model.add(Activation('relu'))

model.add(Dropout(0.5))

model.add(Dense(num\_classes))

model.add(Activation('softmax'))

# initiate RMSprop optimizer and configure some parameters

opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)

# Let's create our model

model.compile(loss = 'categorical\_crossentropy',

optimizer = opt,

metrics = ['accuracy'])

print(model.summary())

**Visualizing the Model:**

# Save our model diagrams to this path

model\_diagrams\_path = '/home/deeplearningcv/DeeplearningCV/Trained Models/'

# Generate the plot

plot\_model(model, to\_file = model\_diagrams\_path + 'model\_plot.png',

show\_shapes = True,

show\_layer\_names = True)

# Show the plot here

img = mpimg.imread(model\_diagrams\_path + 'model\_plot.png')

plt.figure(figsize=(30,15))

imgplot = plt.imshow(img)

**ModelCheckpoint, EarlyStopping, ReduceLROnPlateau:**

from keras.optimizers import RMSprop, SGD

from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau

checkpoint = ModelCheckpoint("fruits\_fresh\_cnn\_1.h5",

monitor="val\_loss",

mode="min",

save\_best\_only = True,

verbose=1)

earlystop = EarlyStopping(monitor = 'val\_loss',

min\_delta = 0,

patience = 3,

verbose = 1,

restore\_best\_weights = True)

reduce\_lr = ReduceLROnPlateau(monitor = 'val\_loss',

factor = 0.2,

patience = 3,

verbose = 1,

min\_delta = 0.0001)

# we put our call backs into a callback list

callbacks = [earlystop, checkpoint, reduce\_lr]

# We use a very small learning rate

model.compile(loss = 'categorical\_crossentropy',

optimizer = RMSprop(lr = 0.001),

metrics = ['accuracy'])

nb\_train\_samples = 67692

nb\_validation\_samples = 22688

epochs = 10

history = model.fit\_generator(

train\_generator,

steps\_per\_epoch = nb\_train\_samples // batch\_size,

epochs = epochs,

callbacks = callbacks,

validation\_data = validation\_generator,

validation\_steps = nb\_validation\_samples // batch\_size)

**Training the Model:**

history = model.fit(x\_train, y\_train,

batch\_size=batch\_size,

epochs=epochs,

validation\_data=(x\_test, y\_test),

shuffle=True)

model.save("/home/deeplearningcv/DeeplearningCV/Trained Models/cifar\_simple\_cnn\_2.h5")

# Evaluate the performance of our trained model

scores = model.evaluate(x\_test, y\_test, verbose=1)

print('Test loss:', scores[0])

print('Test accuracy:', scores[1])

**Loading the model:**

from keras.models import load\_model

classifier = load\_model('/home/deeplearningcv/DeepLearningCV/Trained Models/8\_mnist\_simple\_cnn\_10\_Epochs.h5')

#### Confusion Matrix and Classification Report:

from sklearn.metrics import classification\_report,confusion\_matrix

import numpy as np

y\_pred = model.predict\_classes(x\_test)

print(classification\_report(np.argmax(y\_test,axis=1), y\_pred))

print(confusion\_matrix(np.argmax(y\_test,axis=1), y\_pred))

### Displaying our misclassified data:

import cv2

import numpy as np

from keras.datasets import mnist

# loads the MNIST dataset

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

# Use numpy to create an array that stores a value of 1 when a misclassification occurs

result = np.absolute(y\_test - y\_pred)

result\_indices = np.nonzero(result > 0)

# Display the indices of mislassifications

print("Indices of misclassifed data are: \n\n" + str(result\_indices))

**Testing data with our Model:**

import cv2

import numpy as np

def draw\_test(name, pred, input\_im):

BLACK = [0,0,0]

expanded\_image = cv2.copyMakeBorder(input\_im, 0, 0, 0, imageL.shape[0] ,cv2.BORDER\_CONSTANT,value=BLACK)

expanded\_image = cv2.cvtColor(expanded\_image, cv2.COLOR\_GRAY2BGR)

cv2.putText(expanded\_image, str(pred), (152, 70) , cv2.FONT\_HERSHEY\_COMPLEX\_SMALL,4, (0,255,0), 2)

cv2.imshow(name, expanded\_image)

for i in range(0,10):

rand = np.random.randint(0,len(x\_test))

input\_im = x\_test[rand]

imageL = cv2.resize(input\_im, None, fx=4, fy=4, interpolation = cv2.INTER\_CUBIC)

input\_im = input\_im.reshape(1,28,28,1)

## Get Prediction

res = str(classifier.predict\_classes(input\_im, 1, verbose = 0)[0])

draw\_test("Prediction", res, imageL)

cv2.waitKey(0)

cv2.destroyAllWindows()

**Plotting the accuracy:**

# Plotting our loss charts

import matplotlib.pyplot as plt

history\_dict = history.history

loss\_values = history\_dict['loss']

val\_loss\_values = history\_dict['val\_loss']

epochs = range(1, len(loss\_values) + 1)

line1 = plt.plot(epochs, val\_loss\_values, label='Validation/Test Loss')

line2 = plt.plot(epochs, loss\_values, label='Training Loss')

plt.setp(line1, linewidth=2.0, marker = '+', markersize=10.0)

plt.setp(line2, linewidth=2.0, marker = '4', markersize=10.0)

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.grid(True)

plt.legend()

plt.show()

**Run with OpeCV:**

import cv2

import numpy as np

from keras.models import load\_model

img\_row, img\_height, img\_depth = 32,32,3

classifier = load\_model('/home/deeplearningcv/DeepLearningCV/Trained Models/cifar\_simple\_cnn.h5')

color = True

scale = 8

def draw\_test(name, res, input\_im, scale, img\_row, img\_height):

BLACK = [0,0,0]

res = int(res)

if res == 0:

pred = "airplane"

if res == 1:

pred = "automobile"

if res == 2:

pred = "bird"

if res == 3:

pred = "cat"

if res == 4:

pred = "deer"

if res == 5:

pred = "dog"

if res == 6:

pred = "frog"

if res == 7:

pred = "horse"

if res == 8:

pred = "ship"

if res == 9:

pred = "truck"

expanded\_image = cv2.copyMakeBorder(input\_im, 0, 0, 0, imageL.shape[0]\*2 ,cv2.BORDER\_CONSTANT,value=BLACK)

if color == False:

expanded\_image = cv2.cvtColor(expanded\_image, cv2.COLOR\_GRAY2BGR)

cv2.putText(expanded\_image, str(pred), (300, 80) , cv2.FONT\_HERSHEY\_COMPLEX\_SMALL,3, (0,255,0), 2)

cv2.imshow(name, expanded\_image)

for i in range(0,10):

rand = np.random.randint(0,len(x\_test))

input\_im = x\_test[rand]

imageL = cv2.resize(input\_im, None, fx=scale, fy=scale, interpolation = cv2.INTER\_CUBIC)

input\_im = input\_im.reshape(1,img\_row, img\_height, img\_depth)

## Get Prediction

res = str(classifier.predict\_classes(input\_im, 1, verbose = 0)[0])

draw\_test("Prediction", res, imageL, scale, img\_row, img\_height)

cv2.waitKey(0)

cv2.destroyAllWindows()